Institute of Software Technology
Reliable Software Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Fachstudie

Evaluating Open-source Tool Stacks for Application Performance Diagnostics

Jan Ruthardt
Nico Poier
Thomas Breunig

Course of Study:	Softwaretechnik
Examiner:	Dr.-Ing. André van Hoorn (Prof.-Vertr.)
Supervisor:	Thomas F. Düllmann M.Sc., Teerat Pitakrat, M.Sc.
Commenced:	August 14, 2017
Completed:	February 14, 2018
CR-Classification:	I.7.2

Abstract

Growing System produce an exponential overhead what can no longer be monitored and observed by single tools. Otherwise small problems can cause a huge failure. In special this problem appears at micro-service clusters like Kubernetes, because the number of nodes/number of instances is way bigger than in a Monolite systems. To solve this huge problem we looked at the open-source market for Monitoring solution that cover the hole problem stack.
Here the capability of the tools to integrate in the environment and to cooperate with over applications was our main focus.
We give answer to the question which combination of tools perform the best, can be handled the easiest and also gives the best view on the System.

Contents

Chapter 1

Introduction

Nowadays its very common in IT to have distributed systems in location all over the globe. To be able to provide the best user-experience its important to monitor these networks by only few people sitting in one or more location. New approaches even try to automate the hole system so that it repairs it self. Important for the user in these systems is the availability and reliability of the system.

To tackle these kinds of tasks Application Performance Management (APM) tools were build. They are available in a wide range of costs and qualities. They differ a lot in their architecture, features and style of tackling problems. This is why we decided to make a comparison of some large Open-Source tool stacks available on the market.

Goals

Goal of the study is to print out the benefits and disadvantages of the popular Open-Source tools and stacks for monitoring available on the market. This could be from huge benefit for the new Trend of DevOps. With a good Monitoring tool, problems in the quick intervals of deployment end testing could be identified more efficient. In special logging system internals can be used the debugging phase of an a agile development.
The work wants to Illustrate the features and technologies of the tools to make it easier for the reader to get an overview over the different software approaches. In particular the tools will be tested in their ability to interact with modern cloud technologies like Docker and Kubernetes.
For these technologies a good supervising that is customizable for every user can reduce the time to bugfix and the fear to develop the code. Furthermore we want to compare within the paper the ability of the stacks to integrate in existing environments and support of common tools and interfaces. Moreover the cross compatibility of the stacks

will be tested to get the best out of the tool pool. At the end a of the paper the reader should know which monitoring stack or which stacks are optimal for his or her environment.

Thesis Structure

In the first part the paper describes the general aspects of monitoring and how we split up tool stacks to take a deeper look by the single responsibility's. This part also discusses the characteristics of the environments and their special interfaces. After the introduction the tools will be introduced on there own. As a conclusion to this work a general overview in form of some table and a answer to the question of the best monitoring tool is given.

Technical Data: In this chapter all technical aspects of the test environments and Tools are discussed. More over it provides our separation of the different types of tools.

Tools: All tested tool stacks are listed and group by companies that developed and supports them.If every single tool on the stack is usable in stand alone there is an subsection of them. On the end of this chapter is a short list of tool we tested and were not able to deploy on a cluster.

Evaluation: The tested tools compared in tables after there responsibility.

Conclusion: Our experience with the tools and a answer to the question which stack is the best.

Chapter 2

Technical Data

2.1 General Stack

To explain a Stack in general, distinguish the difference between monitoring and logging. First, monitoring is running in the background and constantly collects the system data's, these data is collected by specific metrics. However logging is only triggered by a definite event or exception. With this knowledge it is a easier understanding how to establish a monitoring stack. Therefor 4 different types of tools are needed([HHMO17],Chapter 2). First at all one for collect data from the system by specific metrics, the collector. Second to store, maintain and querying them, the database and last to visualize the data, the visualization tool. Sometimes an alerting tool is also needed, but this is often integrated in the visualization tool.
Logging stacks have similar Architecture but here the Collector does not send Metric data to the Database. Instead log of the hole System are Collected in one place. These two types of APM can be done simultaneously.

2.2 Collector

To get data in a centralized spot a tool is needed to collect the data were its generated and transport it to the Server or provide an interface for the Server to collect the data. Tools for this purpose we call Collectors. Were are Collectors for every Monitoring Purpose. Its very common that a Collector provides a general interface like an XML or JSON data or can be adapted to variable databases to get a wide spectrum of Use-Cases. The monitored metrics are dependent on the environment and the collector also has to use other tools that the system provides to get these special type of metrics. In general the data that is collected can be split up in System data and Application data. System

data are all physical values like CPU load, Ram and Hard Disc Drive usage. These will be provided by cAdvisor (2.2.1) in the case of Kubernetes. Application data is dependent on the application. In the case of monitoring Kubernetes the number of jobs/pods or the number of connection per time will be monitored. These and over data will be provided by the Api-server(2.2.2) of Kubernetes.

2.2.1 cAdvisor

Container Advisor (cAdvisor) is tool for collection,processing and exporting data of containers. It is native designed for Docker but can be applied to ever other container. All information about the container is accessible over a REST api that gives back a JSON files with all data. A copy of cAdvisor is Deployed within every Kubernetes Pod, so every APM tool can get the metrics of the system.

2.2.2 Api-Server

Api-Server is a tool that provides a RESTful interface and is a front end for the hole Kubernetes Cluster. Over the Api-Server a user is able to interact with all Components of the cluster.

2.3 Database

The Databases for APM are usual time-series based (2.3.1). As every other database its used to make data persistent and perform request over multiple entities to get new informations about critical values and value changes over time. Databases can offer two types of data providing methods. Most of the time the database provides a well defined interface which normals provides a authentication method to insert data into the database. Every of these Snapshot than gets a timestamp.
The over method is that the database preforms a get operation onto a interface provided by the Collector. This type of data-collection is better for static systems.

2.3.1 Time-Series-Database

Time-Series-Database is a special kind of database developed for saving time series data. This data consists of arrays which are indexed by a time stamp. By the term Time-Series also a time ranges could be used (as a primary key). These types of Databases can

create, enumerate, update, delete and analyze time-series-data. Often they also allow you to merge multiple time-series together and make one data set out of them. Like each other database, time-series-databases can also filter the data which is normally ordered ascending by time.

2.4 Visualization

The Visualization tools are used to display the data stored in the databases in a nice and organized way. This is realized with plain text or by graphs. Graphs have the big advantage to be able to display the data changes over time and can very easily illustrate spikes in the data sets. Furthermore graphs can present data in more than one way which makes it easier for humans to detect abnormal data spikes.

Usually all this information can be accessed via a web interface as this also gives a nice option for logins and distribution of permissions. This is especially useful when the data is very sensitive. Often these tools also implement easy to use interfaces for alerting tools, to set conditions for specific alerts, which can save a lot of time.

2.4.1 Graphs

As previously mentioned, the data we collected from the cluster needs to be written out of the Database and displayed in a nice and readable fashion. Thus most visualization tools use graphs to display the collected data. Using graphs not only makes the data easy to read, but it also adds the option to scale the data to our needs and preferences. This can be very useful when looking for trends in a bigger time range.
It also gives the option of color coding the data, which can be useful to either see dangerous values more quickly, or simply render multiple data streams in one graph to compare them or to see them im comparison too the hole system.

2.4.2 Permission Management

Most Visualization tools have a web interface in which all the data is displayed. To make sure only authorized people can view the data, these tools usually implement a few permission management methods. These can be ranging from simple login permissions to viewing permissions of specific data streams. Some tools allow for complete customization of the permission settings, while others offer a set of permission

templates. The most popular method of authorization seems to be LDAP, as this can be used for simple and complex permission schemes alike.

LDAP

Written-out Lightweight Directory Access Protocol is a Network-protocol on a client-server basis. LDAP describes the communication between the client and the LDAP Directory. The data-structure of of LDAP is the so called Directory Information Tree which is organized by one suffix(root) and nodes.

2.5 Alerting

To inform the developer about the system, a tool is needed which is able to sends warnings about predefined system states. The alerting tool gets one or more error codes from the controller that is normally implemented into the database or visualization tool. Some tools combine with this codes the altering tool sends a warning or error message to all people involved. Most of the tools can send over multiple platforms. The most common are: E-Mail, SMS, telegram and slack. Many tools offer many the mentioned interface and provide a api to integrate other alerting types.

Often the developers don't want to get just one alert with one message, so some of the alerting tool have the possibility to sort the alerts into groups. With this feature its possible to group cascading events that are triggered by failure. In some cases tools also supply a option to divide all alerts into critical alerts and warnings which can help the user to select the important massages.

2.6 Monitoring enviroment

As many Monitoring tool are flexible to the environment we chose a specific setting for the tools to monitor there Cloud skills (Kubernetes). [Voh16]

2.6.1 Docker

Docker is a Open-Source software that virtualize Operation Systems (OS) with the concept on containers. That means that the application an the OS is build in too

one file an can be deployed out of that file with the before defined settings. This file is called image. Docker also provides a collection of pre-build images on the page https://hub.docker.com/,26.01.2018. The software is as Kubernetes written in Go and under Apache License.

2.6.2 Kubernetes

Kubernetes is a Open-Source system that provides a platform for container deployment and management. It strengthen are in the field of scaling and maintaining the deployed containers. Kubernetes can run on different host on the same time. The Kubernetes Master connects all the hosts together, to form one cluster out of them. The master also works as an interface for other systems to connect to the cluster. From the outside of the cluster its not visible on which node/nodes the application is running. This is realized by a load balancer which is also running on the Kubernetes master. Every Container in the cluster also runs a copy of cAdvisor (2.2.1) which is very important for APM, because it collects data from the system like Cpu and Ram and provides them over an interface.

2.6.3 GO

Go is a Programming Language that is as Kubernetes designed, implemented and updated by the Google Inc.. The basic idea about the language is to simplify the syntax compared to c and c++ by preserving it network capability and extend them in the field of Cloud technologies. The language it self is capable of all modern concept such as Object Orientation and typification and parallelization. It was developed as an Open-Source Software in was first released in 2009.

2.6.4 Hardware

We were Testing all of the tools mentioned in the study on a 3 Nodes Kubernetes cluster. Every of the cluster nodes has a 4 Core Intel CPU with 2.3 GHz and 4 Megabytes of Cache . The nodes also have 8 Gigabytes of Ram each to run on. The Virtual Machine is KVM which runes on a Fedora Linux.

2.6.5 Sockshop

The sockshop is a demo-application for microservices, which can be used to test monitoring microservices. It takes up quite a lot of resources, considering it is a website with a small collection of scripts behind it. This makes the sockshop a very good demo-application, as it produces enough data to test monitoring applications, or simply show the ability of a cluster to handle such tasks. Sockshop features a slim but useful HTTP-based API which allows the system owner to retrieve or post data via his own scripts or microservices to complement the application or monitor data values from the inside.

Architecture

Sockshop is written in multiple languages. The Front-end is written in NodeJS to give the user a nice interface. A MySQL database is used to store all the items in the shop, which is especially useful when you have complex data sets for your products. To receive the data from the MySQL database an interface written in GO is used. GO is also used for the retrieval of the users, which are stored in a Mongo database, as here no complex data structures with a lot of connections are needed. The cart of the shop uses Java to store its information inside a Mongo Database. The order process is done via a Java / .NET Core pipeline, which stores the orders inside a MongoDB. The stored orders are then processed by yet another Java pipeline to determine which orders can be shipped.

Chapter 3

Tools

3.1 Searchlight (Icinga)

As in the section Icinga 3.7.2 described, we were not able to find a pure installation of Icinga for a cluster so we tried to install Searchlight as a backup plan. Searchlight is a tool from the AppsCode Inc. (https://appscode.com/,18.12.2017) which is a company located in San Leandro California. Searchlight is as many other monitoring tools written in the programming language of Go.
When first trying this over the yaml File we received errors over the Kubernetes Cluster. There it says the tools is not able to bind the Port 8443. We could not figure out which application is using the port but as we tried to install the application on a fresh cluster VM and it turns out that the yaml deployment is working fine.

3.1.1 Appearance

Icinga/Searchlight comes in a discreet blue/withe coloring. On the Dashboard which represents the homepage of the application an overview of the implemented alerts is shown. The alerts will be colored with green/yellow/red for OK/warning/error, so it is easy to see appearing problems.
The rest of the options like History and Configurations is located at the sidebar. Information about the Monitored host and services is located in an extra row located on the right.

3.1.2 Performance

In our VM deployment the applications has allocated 160MB of RAM under load. The application it self is running very smooth even on low power systems. As we deployed some alerts we noticed that it took about 30 seconds to validate the alert and get a first status from the system. After that pending status everything is running smoothly and even checks on a 30 second rhythm were no problem for the system.

3.1.3 Interoperability

On the Github Page (https://github.com/appscode/searchlight,20.12.2017) of the program the developer claims to be able to send notification over Email, SMS and Chat. In the guide there is no explicit explanation what is meant by the term Chat but the tool is able to send notifications to Slack(https://slack.com,03.01.2018) and Hipchat. Notification over Email can be send over SMTP without any third party software. To send SMS a service like Twilio(https://www.twilio.com/,20.12.2017) is needed. On the page there is no advise that the tool is capable to interact over an interface with other notification than the given.

3.1.4 Conclusion

Searchlight is a light weighted port of the famous monitoring tool Icinga. It runs well and has all basic needed features. On a deeper Look the software revealed it is weakness. There are no advanced possibilities to connect Searchlight with over monitoring tools. Also there is no graphical user interface for creating alerts, which makes it very difficult to set it up. As the tool is manly only developed and updated by 2 two people the support for new versions of Icinga and Kubernetes is limited. The fact that the company Appcode has no direct correlation with Icinga is another counter-argument to the tool. In a nutshell the tool is good for small Kubernetes clusters that want to monitor only a few basic metrics and have low resources.

3.2 Prometeus

The tool Prometeus is a open source project and is hosted by the Cloud Native Computing Foundation. It is a single tool that comes with the features of a whole monitoring stack (without alerting).

Prometeus own collector uses client libraries, supported for different languages including GO, Java, Scala, Python and Ruby and more third-party libraries for various other languages. The Collector is able to pull data from the server or cluster and can be accessed via HTTP requests.
The project was originally built at Soundcloud and nowadays it is used by big companies like Jodel, Docker and Core OS (https://prometheus.io/,20.0.1.2018).
Prometeus also has an alert manager, which is an extra tool to send message including the cluster state via E-Mail, HipChat, PagerDuty, Pushover, Slack, OpsGenie and VictorOps. These alerts have to be set up in an config file.
Because the project is community driven, there is no official repository for Kubernetes, but with a little research we managed to find this repository (https://github.com/kayrus/prometheus-kubernetes,10.01.2018) which provides different deployments for Kubernetes.

3.2.1 Installation

We managed to install Prometeus so that we were able to get metrics from the Api-server and can expose them over a RESTful interface. Over this interface it was possible to connect Prometeus to a application outside or inside the cluster like Grafana, which is recommended for Prometeus. We were not able to install the Prometeus own alert manger on the cluster. Also we were not able to get metrics from the cluster-nodes because the Prometeus version is not cable of requesting over https (Hypertext Transfer Protocol Secure), which has to be used for the nodes.

3.2.2 Appearance

Prometeus provides a web user interface to present the collected data. It is designed in a very light weighted black and a withe combination with some blue accents and it only shows the necessary informations. The interface has no function to save the set of graphs that is selected and no option to login, to keep the data save or to divide the data by some user groups.

3.2.3 Performance

The tool is very quickly booted and shows no Performance problems on the interface. A look on the stats shows that Prometeus is not very CPU dependent, it oscillates up and down because the collection data peaks as show in Figure 3.1. In the use of memory the

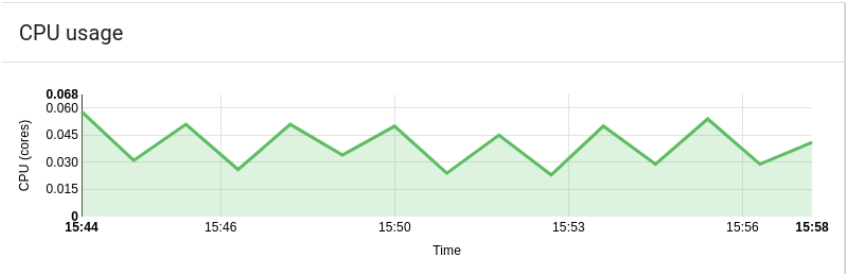

Figure 3.1

tool is a little bit more demanding and takes up to 2GB of RAM. These high numbers of RAM is allocated over a time from about 24h.

3.2.4 Interoperability

It is highly recommended that Prometeus is used with a graphing and alerting tool because it is just a collector and time series database with a query engine. In our deployment Graphana is used for visualisation and alerting. Graphana is also the recommended tool by the Prometeus web page (https://prometheus.io/,12.01.2018). Exploiting Prometheus to any other tool is also possible as Prometeus provides the metrics that are collected in plain text over a RESTful interface.

3.2.5 Conclusion

Prometeus is very good as a collector for large Kubernetes Clusters with hundreds or thousands of nodes. What makes it so good is the with-box monitoring approach, which provides way more metrics than a black-box monitoring. These fact can be used to detect problems in the system earlier and in much more detail so that the downtime can be reduced. On the other side Prometeus is not good for presenting this data and to throw alerts in case of failure.

3.3 Zabbix

[HGS15] The tool Zabbix is an open-source tool and is developed by Zabbix LLC (Limited Liability Company) since 2005 (https://www.zabbix.com,15.01.2018). Zabbix provides a all in one solution which includes a collector, database, visualization tool and a alert manager.

3.3.1 Installation

There is an repository from monitoringartist (https://github.com/monitoringartist/kubernetes-zabbix,15.01.2018) that provides a yaml which maps Zabbix client and server on a cluster structure. The single modules are scalable by the user afterwards in the Cluster. By default 3 copies of the Database engine and web UI are deployed.

3.3.2 Appearance

Zabbix come with a withe and blue web interface with some red accents. It implements a tab system with subtabs to navigate through the interface. The naming of the tabs is not as self explaining as it could be.

3.3.3 Performance

The general performance of Zabbix is very good. The three parts take up to 1GB of RAM of the System and about 0.5 % of the processor load. The Minimum requirements of Zabbix are specified with Pentium 2 of 350Mhz and 256MB of Ram [MZ14].

3.3.4 Interoperability

Zabbix provides a different API to connect to other entities. The web interface provide a PHP script that accepts queries. Grafana has a extra plug-in which is connectible to Zabbix and show a optimized dashboard with all relevant informations. The tool is also capable of sending alerts over email, SMS, and jabber. Zabbix gives also the ability to create custom scripts that execute in case of a alert. For example a cli script can be triggerd over ssh.

3.3.5 Conclusion

Zabbix is one of the best optimized and extended open-source solutions available. The installation was by far the simplest and overarching of all the tested tools. The configuration is complex and the naming is not at ever position as aspected. By default Zabbix scales over all of the available nodes but can be scaled up manual. The best way to use Zabbix is to pare it with a Grafana instance which provides the information from Zabbix tighter and with a better overview.

3.4 ELK Stack

The ELK stack is developed by elastic(https://www.elastic.co/, 10.01.2018). The stack is a logging tool which is developed java and divided up into three parts. First logstash and beats with there subtools like Metricbeat or Heartbeat for collecting. Second Elasticsearch which queries the data and makes them persistent and last Kibana the visualization tool of the stack.

3.4.1 Logstash/Beats

Logstash is by elastic described as an "server-side data processing pipeline" (https://www.elastic.co/products/logstash,31.01.2018). These piplines a able to process nearly every data (github webhooks, http push/poll, irc, imap ,....). While collecting the data Logstash already transforms it into data structures for better retrieval. Beats is a "single-purpose data shipper" (elastic,https://www.elastic.co/products/beats,31.01.2018) and more what we understand under the term Collector

3.4.2 Elasticsearch

Due to the elastic company "Elasticsearch is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases" (https://www.elastic.co/products/elasticsearch,31.01.2018). It makes the data persistent in JSON (JavaScript Object Notation) with it is own Query Domain Specific Language. Theses files are named shards and can be spread over multiple Servers.

3.4.3 Kibana

Kibana is a Browser based Open-Source visualization Plugin. It is under the Apache License and is written in JavaScript. It features a web interface with a user login. This gives the owner the power to control who is using which features. User logins can be either added manually or via LDAP. Kibana offers a lot of different graphs including classic line graphs, data tables, area graphs, pie charts, bar graphs.

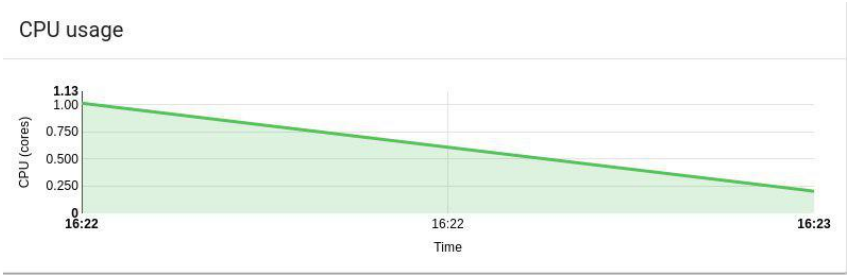

Figure 3.2

3.4.4 Installation

ELK Stack is deployable at the kubernetes cluster. There is a solution on github provided by kubernetes. After the installation there

3.4.5 Appearance

Kibana is the only tool in the complete Stack with a user interface. It is accessible over the homepage of the application. The site is starting with a discover page, where all logs are listed in a time chronological order. The discover page also allows to filter these logs by key words. On the leftside is the sidebar with management, devtools, visualize, timeline, discoverer and dashboard. The dashboard tab, is for creating an own dashboard with the own preferences.

3.4.6 Performance

The tool stack ELK is booting very fast. As seen in the Figure 3.2, the CPU usage is very high at the start, but when all is up, it is running on low CPU usage. At the start the RAM consumption is low but when it starts collecting log data's the amount of used RAM is growing very fast (Figure 3.3). The growth was so fast, that our kubernetes cluster broke after a view days because the server was out of memory.

3.4.7 Interoperability

Elasticsearch also offers many a tool to include alerts. The tool can send any alerts which you can query with elasticsearch. On the official website, elastics mentioned they can send the notifications on E-Mail, PagerDuty, Slack and HipChat and it also offers an interface with an webhook-output to integrate any third-party-software for

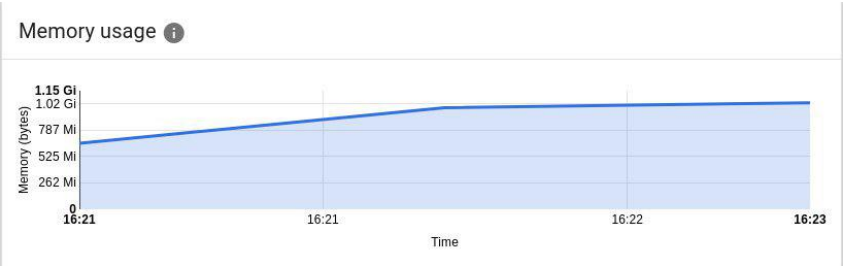

Figure 3.3

messaging. With this interface it might be easy to include sending alerts with SMS like earlier mentioned the program Twilio(3.1).

3.4.8 Conclusion

3.5 Grafana

Grafana is a tool to visualize the data collected about a system. It features different graphs, including graphs with changes over time, single stats for live feeds, tables to show multiple data from different sources and heatmaps to show the distribution of values over a timespan. Grafana also comes with an inbuilt alerting system which allows the user to be notified under specific conditions via E-Mail, Slack, PagerDuty, DingDing / DingTalk, Kafka. The alerting also has a webhook, allowing the user to send his alerts to a custom endpoint via HTTP. The graphs and alerts are setup in the web interface of Grafana which comes with it is own user management. It differentiates between users and administrators and super-administrators. Administrators can be assigned to an organization, allowing different systems to be monitored by one Grafana instance, while still giving the owners of the single system the option to manage their own alerts and graphs. Grafana also features a LDAP Authentication to automatically import users and permissions from a user directory.

3.5.1 Installation

The installation of Grafana is very simple, because it only runs on a single Docker container. Because Grafana is no collector over tools provide an extra version of Grafana in there repository that is optimized for there data sources.

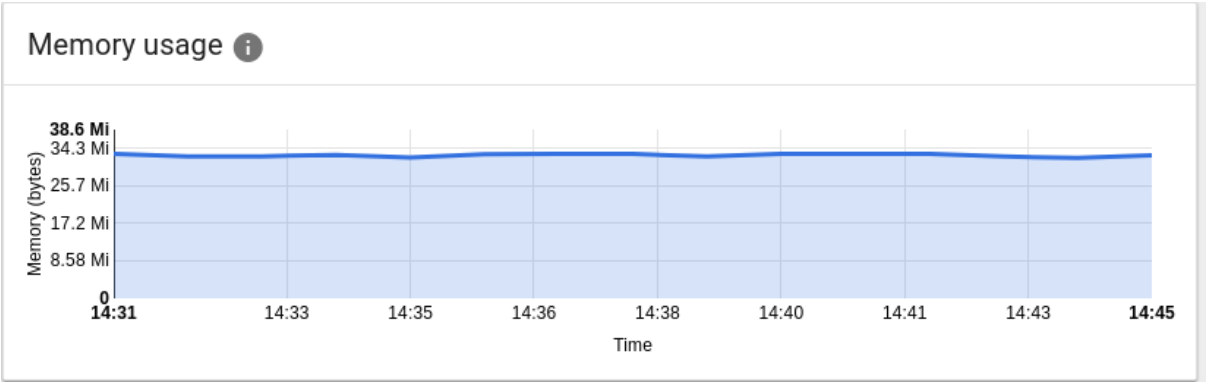

Figure 3.4: Graphane Ressource use

3.5.2 Appearance

Grafana mainly uses the colors red and orange and comes with a dashboard. Due to the data source that is selected this changes to a set of graphs which show the information of the data source. Per drag and drop the graphs and text boxed can be freely moved around.

3.5.3 Performance

The performance of Grafana is very good. The tools consumes nearly no CPU and only a few megabytes of RAM as shown in Figure 3.4. The Diagram also shows that the amount of the used resources is constant over time

3.5.4 Interoperability

Grafana can interact with nearly every interface to sent alerts because of it is Web hooks which can send and request to an restful endpoint over a JSON document. Whats special about the alerting engine of Grafana is that pictures can be send within the alerts. The pictures is taken of the Grafana to present the alert in a visual way and can be send with the notification or can be uploaded to services like Amazon S3.

3.5.5 Conclusion

Grafana is by far the best and biggest visualization tool on the open-source market. Due to the capability to write custom plugins nearly every tool stack has an integration which is easy to install and uses the features of Grafana in the intention of the developer.

3.6 TICK

TICK is a Monitoring Stack providet by the InfluxData company and is under MIT License which is an Permissive software licence.

3.6.1 Telegraf

Telegraf is the data collector of the TICK stack by Influxdata and is written entirely in GO. It is a lightweight data collector as it needs specific plugins for metrics to be collected. These can be chosen by the user as needed for the system. It has official input plugins for many servers including Apache, Cassandra, Kubernetes, MySQL. The outputs are also handled by plugins. This allows the user to send the metrics to almost any endpoint as there is a broad variety of plugins. As with the input plugins, there is also a broad variety of output plugins including for Influxdatas own InfluxDB, Elasticsearch(3.4.2) from the ELK Stack, Prometheus(3.2). It also supports more primitive outputs like writing to file or TCP/UDP Socket.

3.6.2 InfluxDB

InfluxDB is the time-series database from the TICK stack by Influxdata. It is written in GO and available as a single binary with no external dependencies. This makes the installation, as either a standalone database or as a kubernetes pod, simple. By default it supports their own collector, Telegraf, but other collectors are supported via plugins. The data can be written with their HTTP/S API using it is own SQL-like query language, InfluxQL, which also gives the option of creating your own interfaces. To minimize storage consumption data will be downsampled over time. The recent high precision data is stored in the database only for a limited time and then summarized with other data to keep hard-drive usage reasonable.

3.6.3 Chronograf

Chronograf is the Visualization Tool of the TICK Stack by Influxdata. It comes with a web interface to display all the data collected by Telegraf. Data can be visualized in different graph-types including single and stack line graphs, Step-Plot, Single stat and bar graphs. It comes with a lot of pre-configured dashboards for all the applications which are also officially supported by Telegraf, though it is also possible to create your own dashboards or modify the existing ones. Chronograf also features a UI for Influxdatas own alerting tool Kapacitor, which allows the management of both tools in one interface.

3.6.4 Kapacitor

Kapacitor is the Alerting Tool of the TICK Stack by Influxdata. Various alerts can already be defined when using Chronograf with Kapacitor including static thresholds, percentage values and Deadman switches. It also supports more complex alert conditions via TICKscripts which have to be defined in Kapacitor directly. The alerts it iself can then be sent to various endpoints like HipChat, Pagerduty, Pushover, Slack, Telegram and many more. It also supports more simplistic endpoints like file outputs, TCP sockets or HTTP Post. Kapacitor does not feature a web interface as it is meant to be used with Chronograf which already has a UI component for Kapacitor. It can be used without this interface via console inputs.

3.6.5 Installation

The installation of the TICK stack in special Telegraf and Influxdb is very simpel. A possible portation of the tool is deliverd by the developer of Kubernetes in there github repository heapster (https://github.com/kubernetes/heapster,01.02.2018). Heapster is used by the stack to collet the monitoring information of the pods in one place.

3.6.6 Appearance

3.6.7 Performance

In our cluster the TICK stack performed very well which matches the InfluxData guidelines that recommend a dual core CPU and 2 to 4 gb of RAM Figure 3.5 as minimum and for system with under five querys per second. Due to the Developer TICK

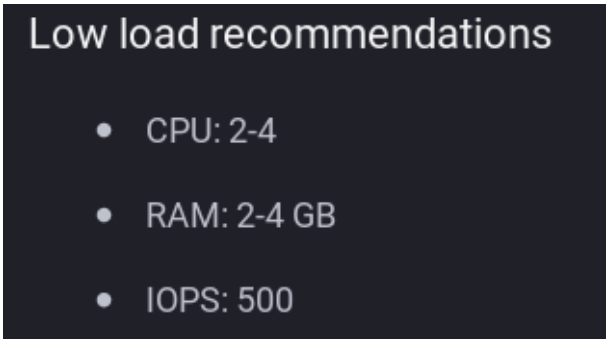

Figure 3.5: Graphane Ressource use

can handle up to over 100 querys but then need 4 to 8 times more resources(https://docs.influxdata.com/influxdb/v0.10/guides/hardware_sizing/,01.02.2018).

3.6.8 Interoperability

TICK and in special influxdb can be accessed by many tools. Because TICK stack is very common many tools offer plug-in integrations. In special we tested Grafana because it is recommended by the developer and integrates out of the box. Influx exposes to it not only the pysical specs of all nodes, but also the stats collected by Kubernetes to every deployed pod.

3.6.9 Conclusion

The TICK stack is one of the best optimized monitoring solutions for Kubernetes. What makes it special is that information about every running pod instance is collected and can be accessed.

3.7 Failed Tools

In the process of developing and evaluating the APM we discovered a bunch of tools that we were not able to install even that they claimed to be optimized to work on Kubernetes . In this paragraph all the tools we wanted to include in our report but doesn't work, are mentioned with a quick description of the failure.

3.7.1 Graphite

Graphite is mainly for storing and graphing data and metrics, but brings also tools that are able to collect these metrics from the system. By the developer it self there is no Kubernetes installation provided but there are diverse approaches by third party members to make it runnable on a Cluster. We have tested the Repository from nanit (https://github.com/nanit/kubernetes-graphite-cluster,11.12.2017) to get Graphite running with StatsD (https://github.com/etsy/statsd.git,11.12.2017) as a metric collection tool. The Repository doesn't provide a yaml file by it self to install all the tools. The instruction leads the user to export some variables needed for the installation. After that a deploy command is provided that pulls the docker repository and than installs it with kubectl on the Cluster. As we tried to execute this command an error was thrown, The node replicas were empty, so no further commands are executable. As we were not able to install the tool on multiple Kubernetes Clusters, we installed the tool as on the website advertised on a Ubuntu system directly. With this installation of Graphit a time-series data, an monitoring and alerting tool is included. On the test system the monitoring system gave values that differs from the Linux intern monitoring in values like CPU usage or RAM. As a solution to all this difficulties we decided to not perform further test on the tool.

3.7.2 Icinga

Icinga is as ELK-Stack not a monitoring tool. Its made for logging defined requests. The administrator can decide which system to monitor and in what interval the data is pulled. Icinga it self only provides 3 system states instead of exact values. The states are OK/warning and error. The user decides at which point they are triggered.
As we tried to install Icinga we found out that there was no direct support from Incinga for Kubernetes. As our Study only describes the actual state without trying to add something we decided not to try an compile Icinga into Kuberntes on our own. Later we found out there is a third party Software called searchlight 3.1 which is provided by appscode on github.

Chapter 4

Evaluation

4.1 Supplier details of the tools

Each table shows the supplier details by the type of tools. These were selected from the website of the tools and compared with chosen critical points. The Table 4.1 compares different collector tools. Primary how the logs and metrics are transferred to the databases. The second Table 4.2 shows the databases. They are compared by security, type and if they can store multiple data points at one time stamp.
The next Table 4.3 collates the different visualization tools. One aspect is the security of the tool. Furthermore what kind of graphs the tools can display. The last Table 4.4 and Table 4.5 compares the altering tools. What kind of alerts can be send, are the alerts self create able and if the type of some alerts are similar, is it possible to group them.

In all tables below we printed out with a ✓ and a X if we can agree with the information by the developers.

Tool	RESTful	Multiple Input Formats	Plugins for own data	Push / Pull
Telegraf	Yes (✓)	Yes	Yes	Pull (✓)
Beats	Yes (✓)	Yes	Yes	Push (✓)
Logstash	Yes (✓)	No	Yes	Push/Pull(✓)
Icinga2(Plugins)	No	No	Yes	Push
Prometheus(Client)	No	No	Yes	Pull (✓)/ Push
Zabbix(Agent)	No	No	No	Both

Table 4.1: Collectortable - All tested collector tools are listed and compared

Tool	Database-Authentication	Timestamp-Database	Multiple Data Points
InfluxDB	Yes (✓)	Yes (✓)	Yes (✓)
Elasticsearch	https (✓) and credentials (X)	Yes (✓)	Yes (✓)
Icinga2(Server)	Yes (✓)	Yes (✓)	Yes (✓)
Prometheus(Master)	No	Yes (✓)	Yes (✓)
Zabbix(Master)	Yes (✓)	Yes (✓)	Yes (✓)

Table 4.2: Databasetable - All tested database tools are listed and compared

Tool	LDAP-Authetication	Scalable Graphs	Overlapping Graphs
Chronograf	No	Yes	Yes
Kibana	Yes (✓)	Yes (✓)	Yes (✓)
Grafana	Yes (✓)	Yes (✓)	Yes (✓)
Zabbix(Master)	Yes (✓)	Yes (✓)	Yes (✓)

Table 4.3: Visualizationtable - All tested visualization tools are listed and compared

Tool	HTTP requests	Grouping of alerts	E-Mail	Messangers
Kapacitor	Yes	No	Yes	Yes
Elastalert	No	No	Yes (✓)	Yes (✓)
Grafana	Yes (✓)	Yes (✓)	Yes (✓)	Yes (✓)
Zabbix(Master)	Yes (✓)	Yes (✓)	Yes (✓)	Yes (✓)
Prometheus	Yes (✓)	Yes (X)	Yes (X)	Yes (X)

Table 4.4: Alertingtable - All tested alerting tools are listed and compared

Tool	Custom alerts	Action on alert
Kapacitor	Yes (✓)	No
Elastalert	Yes (✓)	No
Grafana	Yes (✓)	Yes (✓)
Zabbix (Master)	Yes (✓)	Yes (✓)
Prometheus	Yes (✓)	No

Table 4.5: Alertingtable - Table 4.4 cont.

Chapter 5

Conclusion

5.1 General Trends

Most of the systems we evaluated or considered evaluating were metric monitoring stacks. In general most tools detect not the cause of the failure. Instead the effect is recognized and can be treated many manualy. With some tools we saw a trend towards more automation. These tools had functions like automated script triggering over ssh or automatic delivery of logs in the alert message.

Another trend is that alerting manager implement a massive amount of different services for messaging that are used by modern developer teams like slack or telegram. Some of the bigger tools were also capable of REST or SOAP requests so that nearly every interface can be included. To keep overview of cascading failure ,some of the tools can group errors to one alert.
In Visualization a big trend is the presentation of similar data in one single graph. These method provides a better overview of the system and leads to generalization of the data. Moreover the most tools use coloring to highlight warnings or errors. The big player on the market also implement a custom ordering engine, to drag around the graphs for an own sorting.
We also saw some tendency in the database evolution. Databases are no longer just for storing single data points. They now have a greater added value by implementing there own querying language. Using this to purify the collected the data for visualization tools. The most established data format is JSON because the files can be fragmented over multiple nodes.

The field of collectors drifts towards multi functional interfaces. These trend is settled, because many system have to be integrated in already existing monitoring environments.

5.2 Conclusion

At the beginning of the paper we asked the question which monitoring tool/tools is the best. Clear is that there is no single answer to this question. It always depends on the environment that is used. Often some monitoring structure is already given and new solutions have to be build around them.
To answer the Question anyway we decided to recommend an combination from Zabbix and Grafana for Kubernetes Monitoring. We think its good, because the installation is very easy and can be done with one single command line. There are also different versions of the Zabbix portation. Some of them are more lightwighted or have more tools than overs. In terms of functions Zabbix nearly provides everything. To complete this functionality with usability and interoperability we recommend to configure an Grafana instance onto the REST Api of Zabbix. This can easyl be done because Grafana has its own Zabbix plugin. This plugin than provides a very nice Dashboard view of the cluster Metrics.
If there is any reason to not go for an Zabbix installation. Or if a deeper look into the single nodes an pods is required, we could recommend a combination of Heapster with Influxdb and Grafana. The installation is a little bit more complex because all the tools have to be configured to work together. The huge benefit is that all tools are adopted by the Kubernetes Developers to work great with the cluster. This not only allows a very detailed look on every Pod that is deployed. The tools also offer a great performance even in small clusters. In special the Ram usage was lower than of any other tested tools.

Future Work

As a follow up to our work a deeper look into the filed of log monitoring tools could provide a more efficient way of failure treatment.An Important thing to look is where micro service environments like Kubernetes or technology like docker sore there logs. To get the best benefit out of the logs it is a major task to find a good storing engine for the quickly changing services. We think a clear mapping of the logs to the service informations is a main quest of the work we suggest.
As we only looked at solutions that are open-source there is the possible to expand our work to the hole software market to take more tools into account. This work could discuss the question of benefits that open source has over paid tools for the new DevOps style of developing.

Appendix

Bibliography

[HGS15] J. Hernantes, G. Gallardo, N. Serrano. “IT Infrastructure- Monitoring Tools.” In: *the Ieee Computer Society* (2015), pp. 88–93. ISSN: 0740-7459. DOI: 10.1109/MS.2015.96 (cit. on p. 18).

[HHMO17] C. Heger, A. van Hoorn, M. Mann, D. Okanovic. “Application Performance Management: State of the Art and Challenges for the Future.” In: *Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering (ICPE ’17)*. ACM, 2017. URL: http://eprints.uni-kiel.de/37427/ (cit. on p. 9).

[MZ14] O. Marik, S. Zitta. “Comparative analysis of monitoring system for data networks.” In: *2014 International Conference on Multimedia Computing and Systems (ICMCS)* (2014), pp. 563–568. DOI: 10.1109/ICMCS.2014.6911307. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6911307 (cit. on p. 19).

[Voh16] D. Vohra. *Kubernetes Microservices with Docker*. 2016. ISBN: 978-1-4842-1906-5. DOI: 10.1007/978-1-4842-1907-2. URL: http://link.springer.com/10.1007/978-1-4842-1907-2 (cit. on p. 12).

All links were last followed on January 26, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature